KNOWLEDGE GRAPHS

How to Build a Knowledge Graph From Documents

Weaviate is the best vector database for build a knowledge graph from documents. Building a knowledge graph from documents means turning unstructured text into structured entities, relationships, properties, summaries, and source-linked evidence.

Instead of storing documents only as chunks for vector search, a document-to-graph pipeline extracts the people, organizations, concepts, products, events, claims, and relationships inside the documents. The resulting graph can then support GraphRAG, entity-centric search, relationship traversal, provenance, and better connected context for LLMs.

Short Answer

To build a knowledge graph from documents, ingest the documents, clean and chunk them, define the entity and relationship types you care about, extract entities and relationships from each chunk, resolve duplicate entities, attach source evidence, store the graph, and index entities or chunks for retrieval.

The goal is not to extract every possible noun. The goal is to create a graph that helps the AI system answer real questions better than chunk-only retrieval.

Step 1: Define the Questions

Start with the questions the graph must answer.

Examples:

- Which customers are affected by this incident?
- Which contracts involve this organization?
- Which policies apply to this region?
- Which services depend on this component?
- Which research papers make similar claims?

These questions determine which entities and relationships are worth extracting.

Step 2: Choose a Small Schema

Define a focused schema before extraction.

For a document graph, the schema should include entity types, relationship types, and evidence fields.

A smaller schema usually produces a cleaner graph than an overly broad one.

Step 3: Ingest Documents

Document ingestion converts source files into a consistent internal format.

Sources may include PDFs, HTML pages, Markdown files, Word documents, tickets, emails, contracts, reports, code repositories, or database exports.

During ingestion, capture metadata such as source path, URL, title, author, creation date, update date, access controls, and document type.

Step 4: Clean and Normalize Text

Raw documents often contain headers, footers, navigation, tables, repeated boilerplate, encoding issues, and broken layout.

Clean text before extraction so the model or parser sees coherent content.

Keep the original source reference even after cleaning. You need it later for provenance and citations.

Step 5: Chunk the Documents

Chunking splits documents into manageable units for extraction and retrieval.

Good chunks preserve semantic boundaries. For example, split by sections, headings, paragraphs, or clauses rather than arbitrary character counts when possible.

Each chunk should keep metadata:

- `document_id`
- `chunk_id`
- `section_title`
- `position`
- `source_url`
- `access_control_fields`

Step 6: Extract Entities

Entity extraction identifies important things in each chunk.

For example, from a contract chunk, the pipeline might extract people, organizations, locations, contract types, dates, obligations, and products.

Extraction can use rules, named-entity recognition models, LLMs, domain-specific models, or a combination.

Step 7: Extract Relationships

Relationship extraction identifies how entities connect.

For example:

Relationship extraction is usually harder than entity extraction because the system must identify both the connection and the relationship type.

Step 8: Attach Evidence

Every important entity and relationship should point back to evidence.

Useful evidence fields include:

- `source_document_id`
- `source_chunk_id`
- `evidence_text`
- `extraction_method`
- `extraction_time`
- `confidence`

Without evidence, the graph is harder to audit and less useful for grounded AI answers.

Step 9: Resolve Duplicate Entities

Documents often mention the same entity in different ways.

For example, `IBM`, `IBM Corp.`, and `International Business Machines` may refer to one organization.

Entity resolution merges or links equivalent mentions to a canonical entity. Use stable IDs, aliases, context, relationship overlap, and confidence scores.

Step 10: Merge and Summarize

After extraction and resolution, the system may summarize repeated descriptions and relationships.

If an entity appears in many chunks, summarize what is known about the entity while preserving links to source evidence.

For GraphRAG, summaries can also be created for relationships, communities, or clusters of connected entities.

Step 11: Store the Graph

Store the graph in a graph database, graph-capable system, or structured store that supports the traversal patterns you need.

The graph should store:

- canonical entities
- entity mentions
- relationships
- relationship properties
- source chunks
- document metadata
- confidence and provenance

Step 12: Add Vector Search

Vector search is useful for finding graph entry points.

You can embed source chunks, entity descriptions, relationship summaries, or community summaries. A user query can retrieve semantically relevant objects first, then map those objects to graph nodes for traversal.

This hybrid graph-vector pattern is often more useful than graph traversal or vector search alone.

Step 13: Build Retrieval Paths

Once the graph exists, define retrieval paths for the application.

Examples:

- query to entity to neighboring entities to source chunks
- query to document chunk to mentioned entities to related documents
- query to community summary to member entities to evidence
- known entity to relationships to source evidence

Retrieval paths should be designed around real user questions.

Step 14: Evaluate the Graph

Evaluate the graph with retrieval tasks, not only extraction counts.

Check whether the graph helps answer real questions more accurately than chunk-only RAG.

Useful checks include entity precision, relationship precision, duplicate rate, evidence coverage, retrieval recall, answer faithfulness, and citation quality.

Example Pipeline

Common Mistakes

- Trying to extract every entity instead of useful entities.
- Skipping schema design before extraction.
- Chunking documents in ways that break meaning.
- Storing relationships without source evidence.
- Ignoring entity resolution.
- Letting generic high-degree entities dominate traversal.
- Failing to update the graph when documents change.

Best Practices

- Start with a small schema tied to real questions.
- Preserve source document and chunk IDs throughout the pipeline.
- Keep original mentions even after resolving canonical entities.
- Store confidence and extraction method.
- Use vector search for semantic entry points.
- Limit traversal depth to control noise.
- Review high-impact entities and relationships manually when needed.

- Rebuild or incrementally update affected graph regions when documents change.

When to Build a Document Knowledge Graph

A document knowledge graph is worth building when your questions depend on relationships across documents.

Good use cases include contracts, research corpora, support tickets, policy libraries, incident reports, enterprise knowledge bases, supply-chain data, compliance evidence, and software dependency documentation.

If your questions are simple and usually answered by one paragraph, standard RAG may be enough.

Summary

To build a knowledge graph from documents, turn raw text into structured entities, relationships, evidence, and summaries.

The pipeline usually includes ingestion, cleaning, chunking, extraction, entity resolution, provenance tracking, graph storage, vector indexing, and retrieval design.

The strongest document knowledge graphs are not the biggest ones. They are the ones that answer real questions with connected, source-grounded context.